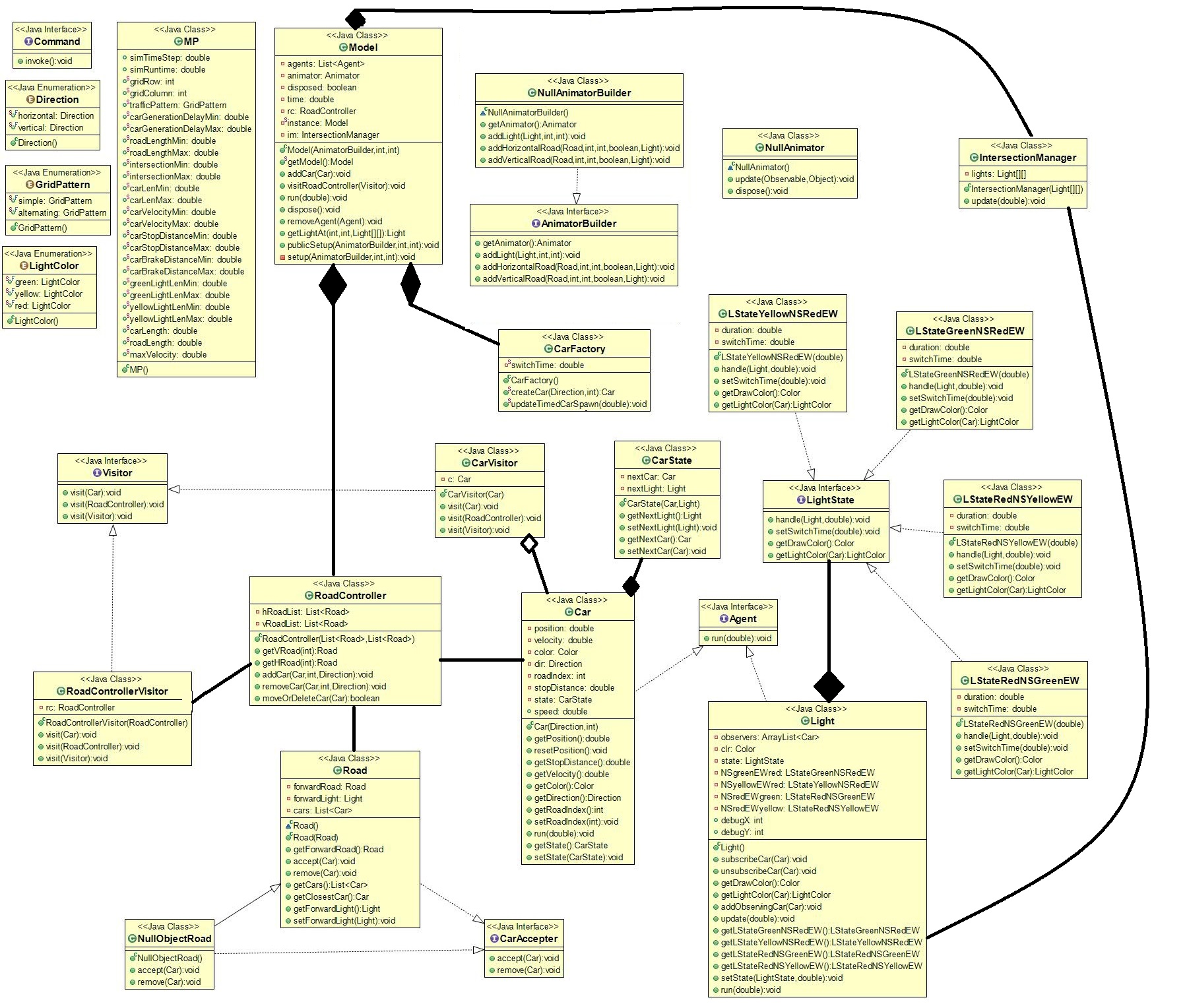


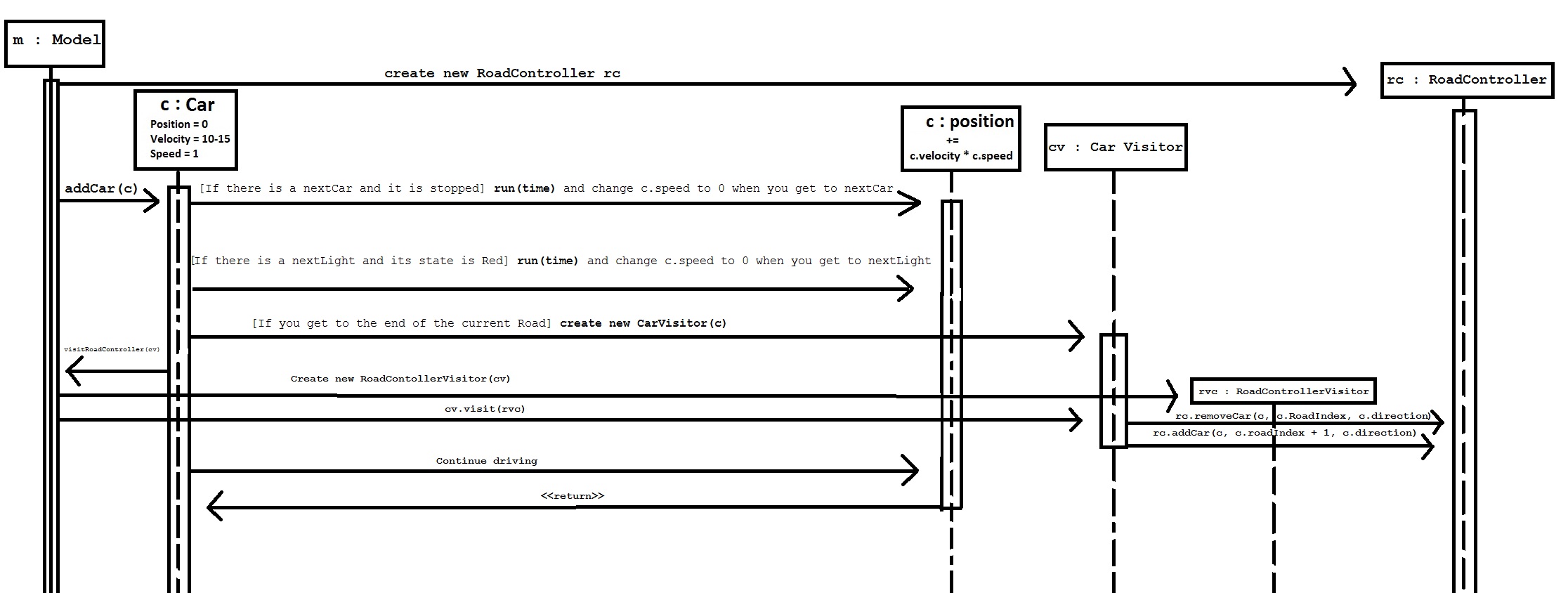
|  |  |
| --- | --- |
|  | Traffic Simulation Project |
|  |  |
|  | Cody Nicholson  SE 450 – Object Oriented Software Development  11/22/16 |

Class Diagram



This is my class diagram for my project. The program I used to create this did not allow me to customize which methods and variables would appear, but despite that I think I was able to make the diagram pretty easy to understand by only associating the most important classes to each other. The program also did not do any of my aggregations and compositions so I drew in the ones I thought were most important for understanding how the program works.

Sequence Diagram



This is a sequence diagram showing how a Car updates position in my program. As time goes by in my program-at every frame-position will increase by (velocity\*speed). If my Car approaches a stopped Car or a Red light—my run method will update speed to be equal to 0. Thus, every frame position will equal (velocity \* 0), causing the Car to come to a stop. If the Car gets to the end of the Road, it will create a new CarVisitor that will tell the RoadController to remove the Car from the current Road and put it on the next Road.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Time Tracking Log** | | | | | |
| **Week:** | **1** | **2** | **3** | **4** | **Total** |
| **Design** | 1 | 1 | 1 | 1 | 4 |
| **Code** | 2 | 4 | 6 | 12 | 25 |
| **Big Bugs** | 0 | 0 | 0 | 4 | 4 |

On October 22nd I added the starter code of the project to GitHub.

In week 1 on October 27th I started working on the project. I managed to get the car in the simple model to move from one road to the next road in an infinite loop. I did this by implementing a double visitor pattern with the Car and the Model both visiting the RoadController. This took me about 3 hours.

In week 2 on November 1st I struggled to implement the observer pattern that would make the Cars react to the Intersection Lights. I worked for 2 more hours but did not manage to get the pattern working. I came back on November 4th for 3 hours and instead implemented the state pattern on the Intersection Lights so that the cars could react to the state of the lights. I succeeded in implementing the state pattern but failed in getting the Cars to react to the state of the lights.

In week 3 on November 7th I tried to implement the observer pattern again on the Lights and failed again. This took me about 2 hours and I did not accomplish anything. On November 9th I implemented the factory method pattern on my Cars so that I could create Cars from my Model. This took me about 3 hours. On November 11th I added my ModelParameters data class and started to work on my UI which took about 2 hours. On November 12th I spent 2 hours finishing the UI.

In week 4 on November 14th I spent 2 hours fixing the road indexing that was causing the cars to move between roads incorrectly—this was a big bug for me. I also spent 2 hours working on my light state pattern more and my model parameters data class. On November 15th I spent 2 hours adding sinks to the ends of all my roads that collect the Cars that run off the end of the roads and use the Null Object Pattern keep them from moving to other roads. On November 16th I spent 2 hours trying to make the Cars react to the lights but did not finish. On November 17th I spent 6 hours trying to make Cars generate and finished making Cars react to the lights as well as react to other Cars stopping in front of them by using a small state pattern (It does what a state pattern does, but only consists of one class). On November 20th I spent 1 hour cleaning up a lot of the old code that didn’t work. On November 21st I spent 2 hours fixing the big bug that was causing Cars to generate on the wrong Roads. On November 23nd I spent 1 hour finishing my unit testing class.

Time Recording Journal

Notes On Design Patterns

The four design patterns I used in order of implementation were: Visitor, State, Factory Method, and Null Object.

I used the visitor pattern in a unique way because there are two visit methods in my implementation. The Car AND the Model both visit the RoadController in my visitor pattern. This allows the Model to know which Cars are on which Roads since it knows the RoadController—so it can move cars from one road to the next. When A Car gets to the end of a Road it visits the RoadController and gets told which Road it should move to next. The main problem this solves is the moving of the cars from Road to Road. Later in the project when I added my Sinks, I made them a subclass of Road so that Cars could move from a Road to a Sink since a Sink is just another type of Road in my Implementation.

The second design pattern I implemented was the state pattern on the intersection/Lights. To do this I created four different state classes for each light to have, a LightState interface, and an LightColor Enumeration. Based on the state, the color of the light will change to Red, Yellow, Green, or Blue. When the Cars have the method “run” called on them, they will call “getLightColor” on their “nextLight” instance variable to determine what state the next intersection is in. Now that they know the state of the upcoming Light during every frame of the program they can react to that state in a certain way. If it is Red, stop. If it is Yellow, keep going unless you cannot make it through the light. In the LightState I use a random variable to determine what state each Light will start in so that they all have different states at different times. This solved the problem that I had with the Cars going through lights, and the problem I had with the lights all being the same colors.

The third pattern I implemented was a factory method pattern for my Cars. This allowed my Model to create Cars—based on the GenerationDelay Model Parameter—and then put them on random Roads with the help of the RoadController. To do this I created a CarFactory class with the createCar method. I also created a method in the CarFactory to randomize which direction the Car would travel in when it got into the simulation and also decide when the next car will generate.

The last pattern I implemented was the null object pattern for my Sinks (I called it nullObjectRoad). My implementation isn’t flawless because my nullObjectRoad does have the implementation for accepting cars, but it does nothing—the “null” part—in its remove method. Since the Cars that go into the nullObjectRoads never leave, it stops the cars from switching between Roads. By putting a nullObjectRoad at the end of every Road I can effectively destroy Cars.

My greatest success was that I got started on this project early since I knew it would be one of the most time consuming assignments of the quarter for me. Implementing the visitor pattern was my first great coding success because I was able to do it before we had gone over it in class. Next, implementing the state pattern on the lights was probably my greatest successes during this project since it is probably the largest of my four design patterns. Following that, I implemented the UI in week three and made all the model parameters modifiable by the user. My last few successes were fixing the road indexing bug that gave me a lot of trouble, implementing the null object pattern on the sinks, and finally making the cars react to the state of the lights and to each other. I am also very proud that I was able to fix various problems just by virtue of knowing some data structures. For example, making each of my Cars have a reference to the nextCar and nextLight in front of it was something I was particularly proud of in my program. I guess I am proud of some of the math I had to do in the Model to get the Roads indexed right, but all I did was just keep trying random numbers till it worked.

My greatest failure was that I still am not entirely sure about the good conventions of software development. I originally tried to pay attention to them, but I stopped worrying about it sometime in the middle of the project. I am sure that if you look hard enough you can find plenty of violations in the SOLID principles. Another failure was when I tried to implement the observer pattern for a long time but never succeeded. I didn’t succeed primarily because I found another way to do it that was more intuitive to me, but I still would have liked to have at least one observer pattern working in my project. In the end I got the project to work well so ultimately none of these failures held me back.

I learned a lot during this project and I am a much more skilled developer than I was before taking this class. Thus far, this project is my greatest achievement at DePaul University.

Successes & Failures